УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
НОВИ САД
Департман за рачунарство и аутоматику
Одсек за рачунарску технику и рачунарске комуникације

ЗАВРШНИ (BACHELOR) РАД

Кандидат: **Лазар Јовановић**
Број индекса: **РА 14-2020**

Тема рада: **Једно решење покретача апликација на бази Андроид отвореног кода**

Ментор рада: **проф. др Илија Башичевић**

Нови Сад, Јул, 2024

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:		
Идентификациони број, **ИБР**:		
Тип документације, **ТД**:		Монографска документација
Тип записа, **ТЗ**:		Текстуални штампани материјал
Врста рада, **ВР**:		Завршни (Bachelor) рад
Аутор, **АУ**:		**Лазар Јовановић**
Ментор, **МН**:		**проф. Др Илија Башичевић**
Наслов рада, **НР**:		**Једно решење покретача апликација на бази Андроид отвореног кода**
Језик публикације, **ЈП**:		Српски / ћирилица
Језик извода, **ЈИ**:		Српски
Земља публиковања, **ЗП**:		Република Србија
Уже географско подручје, **УГП**:		Војводина
Година, **ГО**:		**2024**
Издавач, **ИЗ**:		Ауторски репринт
Место и адреса, **МА**:		Нови Сад; трг Доситеја Обрадовића 6
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)		
Научна област, **НО**:		Електротехника и рачунарство
Научна дисциплина, **НД**:		Рачунарска техника
Предметна одредница/Кqучне речи, **ПО**:		
УДК		
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, **ВН**:		
Извод, **ИЗ**:		**Овај рад има за циљ истраживање и имплементацију покретача апликација на Андроид оперативном систему са корисничким искуством (UX) налик на десктоп рачунар. У раду ће се користити језик Kotlin и алат Jetpack Compose и истражиће се пројектантски и развојни аспекти неопходни за стварање беспрекорног и интуитивног корисничког искуства.** **Кроз комбинацију принципа дизајна усмерених на кориснике и иновативних технологија развоја мобилних апликација, овај рад предлаже нов начин интеракције корисника са њиховим Андроид уређајима.**
Датум прихватања теме, **ДП**:		
Датум одбране, **ДО**:		
Чланови комисије, **КО**:	Председник:	**проф. др Небојша Пјевалица**
	Члан:	**проф. др Мирослав Поповић** — Потпис ментора
	Члан, ментор:	**проф. др Илија Башичевић**

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:			
Identification number, **INO**:			
Document type, **DT**:		Monographic publication	
Type of record, **TR**:		Textual printed material	
Contents code, **CC**:		Bachelor Thesis	
Author, **AU**:		**Lazar Jovanović**	
Mentor, **MN**:		**Ilija Bašičević, PhD**	
Title, **TI**:		**A realization of a Desktop-like Launcher on Android Open Source Project**	
Language of text, **LT**:		Serbian	
Language of abstract, **LA**:		Serbian	
Country of publication, **CP**:		Republic of Serbia	
Locality of publication, **LP**:		Vojvodina	
Publication year, **PY**:		**2024**	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)			
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, **S/KW**:			
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		**This paper aims to explore and implement an user experience (UX) for a desktop like launcher on the Android operating system. Leveraging the power of Kotlin and Jetpack Compose, the study will delve into the design and development aspects necessary to create a seamless and intuitive UX.** **Through the combination of user-centered design principles and innovative mobile application development technologies, this paper proposes a new way users interact with their Android devices.**	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:			
Defended Board, **DB**:	President:	**Nebojša Pjevalica, PhD**	
	Member:	**Miroslav Popović, PhD**	Menthor's sign
	Member, Mentor:	**Ilija Bašičević, PhD**	

Захвалност

Захваљујем се члановима породице и пријатељима на неизмерној подршци током студирања.

Захваљујем се техничком ментору Артему Кузмицком, ментору проф. др Илији Башичевићу као и колегама из тима на стручној помоћи и подршци током израде рада.

САДРЖАЈ

Списак слика

Списак табела

Скраћенице

AOSP	- Android Open Source Project, Андроид пројекат отвореног кода
UX	- User Experience, Корисничко искуство
UI	- User Interface, Кориснички интерфејс
ROM	- Read Only Memory, Меморија само за читање/Системска слика
OEM	- Original Equipment Manufacturer, Произвођач оригиналне опреме

1. Увод

Употреба паметних Андроид телевизора и дигиталних ТВ пријемника је данас у интензивном порасту, омогућавајући гледање садржаја на захтев, коришћење разних апликација и приступ интернету. Традиционални покретачи апликација за паметне телевизоре прављени су са даљинским управљачем у фокусу, који отежава ефективно коришћење веб претраживача, коришћење апликација за продуктивност или персонализовања изгледа према жељама корисника, док су мобилни покретачи често лоше оптимизовани за велике екране.

С обизиром на све већи фокус ових уређаја на мултифунционалност, а с обзиром на проблеме које тренутна решења имају, појављује се потреба за новим типовима покретачких интерфејса који подржавају једноставно и интуитивно коришење ових фунционалности.

Налик-на-десктоп (енгл. desktop-like) покретач апликација решава проблеме традиционалних тв и мобилних покреатача, базирајући се на коцептима радне површине налик оној на персоналним рачунарима као и фокус на управљање помоћу периферија попут миша и тастатуре, што корисницима пружа већу контролу, организацију, и прилагодљивост у коришћењу уређаја.

У овом раду описано је једно решење налик-на-десктоп (енгл. desktop-like) корисничког интерфејса. Решење је реализовано за Андроид платформу са оперативним системом верзије 11.0 или новијим.

Рад се састоји из седам поглавља. У другом поглављу описане су теоријске основе неопходне за разумевање коцепата Андроид платформе, корисничког интерфејса, програмског језика Kotlin као и Jetpack Compose радног оквира. Треће поглавље чини концепт решења у којем су анализирани захтеви потребни за реализацију налик-на-

десктоп (енгл. desktop-like) корисничког интерфејса. Четврто поглавље даје увид у програмско решење са описом модула који су коришћени за израду рада. Пето пофлавље описује тестирање као и сажетак резултата извршеног тестирања. Шетсо поглавље пружа кратак сажетак онога што је урађено у раду и предлог могућих унапређења. Списак коришћене литературе може се наћи у поглављу седам.

2. Теоријске основе

2.1 Android Open Source Project (AOSP)

AOSP (Android Open Source Project) је пројекат који води Google, а који је усмерен на одржавање и развој Андроид оперативног система. AOSP је отворени изворни код, што значи да је код доступан јавности, и свако га може преузети, модификовати и користити за своје потребе. Андроид, као један од најпопуларнијих оперативних система за мобилне уређаје, заснива се на овом пројекту. AOSP садржи све основне компоненте потребне за покретање Андроид уређаја, укључујући језгро базирано на Linux језгру, који управља основним функцијама хардвера, скуп C/C++ библиотека које користе различите Андроид компоненте, виртуелну машину Android Runtime (ART) која покреће Андроид апликације, апликативни оквир који омогућава програмерима приступ API-јевима потребним за развој апликација, те основне апликације као што су Телефон, Контакти и Поставке, које долазе преинсталиране на уређају.

Једна од кључних предности AOSP-а је његова флексибилност. Будући да је отвореног кода, произвођачи уређаја (енгл. OEM) као што су Samsung, Huawei и Xiaomi могу преузети код, прилагодити га и створити властите верзије Андроида које одговарају њиховим хардверским спецификацијама и корисничким потребама. Ово омогућава велику разноликост у Андроид екосистему. Развој AOSP-а је континуирани процес. Google редовно издаје ажурирања која доносе нове функционалности, безбедносне закрпе и оптимизације. AOSP је такође основа за специјализоване верзије Андроида, као што су Android TV, Android Auto и Wear OS. Програмери широм света користе AOSP за развијање прилагођених Android ROM-ова, што омогућава корисницима да имају већу контролу над својим уређајима.

AOSP игра кључну улогу у развоју мобилних технологија. Помаже у стварању заједнице отвореног кода где програмери могу сарађивати, делити идеје и заједно радити на побољшању оперативног система. Кроз AOSP, Google осигурава да Андроид остане динамичан и прилагодљив екосистем способан за брзе иновације. Укратко, AOSP је темељ Андроид оперативног система који омогућава произвођачима и програмерима да створе прилагођена решења и допринесу заједници отвореног кода, истовремено пружајући корисницима широк спектар опција и могућности за прилагодбу њихових мобилних уређаја.

2.2 Корисничка спрега (User Interface – UI)

Корисничка спрега (енгл. User Interface) у развоју Android апликација односи се на визуелни изглед и интерактивне елементе са којима корисници ступају у контакт. У контексту Андроид развоја, корисничка спрега је кључна компонента која одређује како ће апликација изгледати и како ће се корисници кретати кроз њу. Главни циљ дизајна корисничке спреге је да обезбеди интуитиван, естетски пријатан и функционалан интерфејс који омогућава корисницима лако коришћење апликације.

Приликом развијања Андроид апликација, користе се модерни језици и алати попут Kotlin-а и Jetpack Compose-а за креирање компоненти корисничке спреге. Jetpack Compose је модерни радни оквир (енгл. framework) за изградњу корисничких интерфејса који омогућава лакшу и ефикаснију израду елемената корисничке спреге. Кроз декларативни приступ, програмери могу директно описати како интерфејс треба да изгледа и понаша се у зависности од стања апликације, што поједностављује процес развоја и одржавања кода.

Корисничка спрега у Android апликацијама обухвата различите визуелне елементе као што су дугмад, текстуални оквири, слике, иконице, менији и навигационе траке. Ови елементи треба да буду дизајнирани тако да буду конзистентни, приступачни и усклађени са укупним корисничким искуством (UX).

Уз то, важно је обратити пажњу на перформансе корисничке спреге. Апликација мора да реагује брзо на корисничке уносе и да пружа глатке анимације и прелазе. Оптимизација компоненти корисничке спреге и ефикасно управљање ресурсима су кључни аспекти за постизање високе перформансности и стабилности апликације.

Приликом креирања корисничке спреге, такође треба водити рачуна о различитим величинама екрана и резолуцијама које Android уређаји могу имати. Респонзивни

дизајн и коришћење флексибилних дизајна страна (енгл. layout) су од суштинске важности како би се осигурало да апликација пружа добро корисничко искуство на свим уређајима, било да је реч о паметним телефонима, мониторима или другим Android уређајима.

2.3 Kotlin

Kotlin је општенаменски, статички типизиран програмски језик за развој мултиплатформских апликација. Такође је језик који Google препоручује за развој Андроид апликација. Kotlin има пуно одлика модерног програмског језика које га чине привлачним за програмере:

- **Читљивост и концизност**: Сажета синтакса омогућава већу читљивост, брже писање и лакше разумевање написаног кода.
- **Null safety**: Kotlin је дизајниран тако да смањи могућност појављивања Null Pointer Exception грешака. Променљиве које могу имати null вредности су означене приликом декларације и пре коришћења компајлер форсира провере ових променљивих како би смањио број грешака.
- **Корутине**: Kotlin има уграђену подршку за корутине. Корутине представљају "лаке" нити (енгл. lightweight threads) које се користе за извршавање асинхроних операција.
- **Подршка за различите парадигме**: Kotlin подржава и објектно-оријентисану и функционалну парадигму што омогућава писање кода који може да користи класе, наслеђивање или полиморфизам али у исто време одлике функционалне парадигме попут функција вишег реда, типова функција или ламбда функција.

Kotlin се одликује великом флексибилношћу тако да се може користити за било коју врсту развоја софтвера у многим ситуацијама. Разлог је што даје програмеру могућност да бира платформу за коју жели да компајлира код. Неке од платформи које подржава су:

- **JVM** (Java виртуелна машина): Kotlin код може да се компајлира у JVM бајткод, што значи да се Kotlin може користити практично свуда где и Java. Такође Kotlin је 100% интероперабилан са Java кодом, тако да се може користити са постојећим Java библиотекама у тандему.

- **Андроид** : Kotlin је подржан од стране Android Studio алата што омогућава коришћење свих предности Kotlin-а приликом прављења Андроид апликација.
- **JS** (JavaScript): Омогућава писање Kotlin кода који се преводи и компајлира у JavaScript код, што омогућава рад у интернет прегледачу.
- **Нативно** (нативне бинарне датотеке за различите платформе): Компајлира Kotlin код у изворне бинарне датотеке што омогућава писање кода за Linux или iOS платформе.

2.4 JetpackCompose

Jetpack Compose је Google-ов декларативни радни оквир (енгл. framework) отвореног кода намењен за креирање користничког интерфејса на Андроид платформи, те стога погодан за креирање корисничког искуства (енгл. UX) налик на десктоп. Заснован на Kotlin-у Compose нуди декларативни приступ који драстично мења начин на који програмери размишљају о дизајну апликација.

Кључна снага Compose-а лежи у његовој парадигми "шта, а не како". Уместо императивног позиционирања елемената у XML-у, програмери једноставно описују жељени изглед корисничке спреге, а Compose оптимизује рендеровање. У традиционалном Андроид развоју, програмери су се борили са XML дизајном стране и императивном манипулацијом view-ова. Compose то мења својом интуитивном Kotlin синтаксом. Уместо мукотрпног рада са Layout Editor-ом, програмери једноставно описују како корисничка спрега треба да изгледа коришћењем Composable функција, а Compose се брине о имплементацији. Ова декларативна природа смањује количину кода и потенцијалних грешака. Compose-ов реактивни модел базиран на “стањима” је још једна револуционарна карактеристика овог радни оквир (енгл. framework). Када се стање промени, корисничка спрега се аутоматски ажурира, елиминишући непотребни шаблонски код. Један од најузбудљивијих аспеката Compose-а је његова прилагодљивост за разне стилове корисничких спрега, посебно налик-на-десктоп (енгл. desktop-like) корисничко искуство на Андроиду.

Његове напредне компоненте за дизајн стране (енгл. layout), попут LazyColumn за ефикасне листе и ConstraintLayout за комплексне аранжмане, омогућавају програмерима да граде корисничке спреге који се беспрекорно прилагођавају проширеном десктоп моду.

Штавише, Compose-ове Side Effect API-јеве, као што су LaunchedEffect и DisposableEffect, идеално су дизајнирани за управљање ресурсима и асинхроним задацима који су чести приликом дизанирања корисничких интерфејса и искустава.

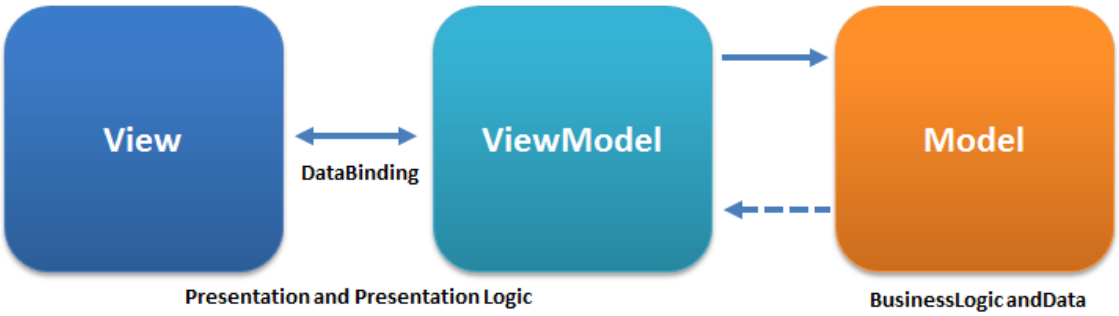
Додатно, Compose-ова подршка за гестове и методе уноса је свеобухватна. На већим екранима, где су пецизне интеракције мишем кључне, Compose нуди софистициране API-еве за детекцију кликова, кликни-и-пусти (енгл. drag-and-drop) операција, па чак и лебдеће (енгл. hover) ефекте.

2.5 Composable функције

Composable функције представљају кључни концепт у модерном развоју корисничких спрега, посебно у оквиру декларативних парадигми програмирања. Ове функције омогућавају модуларан и хијерархијски приступ изградњи корисничке спреге, где се сложене компоненте креирају комбиновањем једноставнијих, самосталних елемената. Свака composable функција дефинише део корисничке спреге и може садржати логику, стање и друге composable функције. Оне функционишу по принципу композиције, где се мање целине комбинују у веће, омогућавајући поновно и ефикасно искоришћење постојећег кода и лакше одржавање. Овај приступ подржава реактивно програмирање, где се промене у стању аутоматски рефлектују на кориснички интерфејс, чиме се постиже конзистентност и побољшава перформансе апликације.

2.6 ViewModel

ViewModel представља срж MVVM (Model-View-ViewModel) архитектурног обрасца, широко примењиваног у развоју апликација помоћу JetpackCompose алата. Он делују као посредник између Model-а (чува и управља подацима) и View-а (приказује податке), чува стања података и посредује у комуникацији између ова два слоја. ViewModel комуницира са Model-ом да би добавио и ажурирао податке, а затим их припрема за приказ у View-у. Оваква архитектура промовише јасну поделу одговорности између слојева, даје подршку за везивање података (енгл. data binding) што олакшава ажурирање корисничке спреге и омогућава писање комплексног и скалабилног кода због лабаве повезаности њених слојева.



Слика 1: MVVM архитектура

3. Концепт решења

3.1 Архитектура решења

“Налик-на-десктоп” корисничка спрега представља комбинацију елемената који су карактеристични за десктоп оперативне системе, као што су Windows или macOS, и као таква има за циљ да пружи корисницима познато окружење које је једноставно и интуитивно за коришћење на Android платформи.

Неки од главних елемената “налик-на-десктоп” спреге су:

- **Палета послова (енлг. Taskbar)** - садржи приказ тренутно отворених апликација као и дугмад за почетни мени, брза подешавања, календар, подешавања звука, брзу претрагу као и taskmenu. Такође мора увек бити присутан на екрану, осим када је нека апликација покренута у пуном екрану (енгл. fullscreen) јер нам она служи као приступна тачка за остале елементе
- **Позадински екран** - приказује пречице апликација, дозвољава њихово померање по екрану и приказује позадинску слику
- **Почетни мени** - приказује све инсталиране апликације на уређају и омогућава брзу претрагу апликација, фајлова и интернета
- **Брза подешавања** - омогућавају приступ основним подешавањима уређаја као и преглед обавештења
- **Контекст мени на десни клик** - пружа додатне опције везане за апликације (постављање и уклањање пречица са екрана, брисање апликација са система) као и пружа приступ активности за персонализацију покретача
- **Taskmenu** - омогућава приказивање и управљање тренутно отвореним апликацијама

Елементи су решени преклапајућим прозорима (енгл. overlay). Ово нам дозвољава истовремен приказ више елемената истовремено, али и могућност да бирамо када ће који елемент бити приказан независно од осталих. Једини елемент који је увек присутан на екрану је палета послова (енгл. taskbar), док су остали преклапајући елементи међусобно дисјунктни, односно може бити приказан само један у датом тренутку.

Када знамо који елементи су нам потребни да би "оживели" наш покретач, следећи корак је дизајн целокупне спреге и њених појединачних елемената. С обзиром да је фокус овог рада на имплементацији дизајна коришћењем алата који су споменути у ранијим поглављима, сам процес дизајна препуштен је колегама из дизајн тима, како је и иначе пракса у индустрији. У наставку ћемо видети њихов предлог дизајна корисничке спреге и њених елемената.

3.2 Предлог дизајна

На сликама 1 и 2 приказан је предлог дизајна за "налик-на-десктоп" покретач апликација. Дизајн је осмишљен тако да буде естетски пријатан као и функционалан, омогућавајући корисницима да се лако сналазе и ефективно користите сам покретач. Елементи које смо поменули пажљиво су распоређени како би обезбедили максималну употребљивост и интуитивност коришћења.

Слика 2: Дизајн корисничке спреге (брза подешавања)

Слика 3: Дизајн корисничке спреге (почетни мени)

Поред самог распореда елемената на слици се могу видети и димензије сваког елемента као и његова удаљеност од других елемената. Ово не само што нам омогућава прецизно постављање елемената по екрану већ нам даје јасне односе између елемената што нам знатно олакшава имплементацију спреге за различите величине екрана. За различите величине екрана апсолутне позиције елемената и њихова величина ће бити различита. Стога позиције и величине елемената израчунавамо на основу дизајна, за једну величину екрана за коју имамо све вредности, као однос висине и ширине екрана и (x,y) координата елемената које су нам дате. Ово нам гарантује тачну позицију и величину елемента на екрану за било коју величину екрана.

4. Програмско решење

У овом поглављу биће представљена програмска реализација корисничке спреге кроз опис функција које је генеришу. Сви елементи су имплементирани помоћу composable функција, које раде на приципу композиције, где се целине описују мањим composable функцијама које се даље комбинују унутар главне composable функције која описује комплетан елемент корисничке спреге. Податке за приказивање добављамо преко ViewModela.

4.1 TeatroosService()

TeatroosService је кључна компонента налик-на-десктоп корисничке спреге која има за циљ креирање и управљање преклапајућим прозорима (енгл. overlay) као и да обезбеди централно место за управљање другим сложеним деловима корисничке спреге. Преклапајући прозори чине темељ за скоро све графичке елементе наше спреге (палету послова, почетни мени, брза подешавања…) јер нам обезбеђује празан прозор , одређене величине и позиције, на који се надограђује сам изглед свих ових компоненти.

Такође овај сервис је задужен за управљање животним циклусом графичких компоненти. Он одлучује када и како приказати или сакрити различите елементе корисничке спреге, осигуравајући критична понашања попут увек присутне палете послова, или међусобну дисјунктност осталих прекривачких прозора.

Као позадински сервис, TeatroosService има могућност константног извршавања, што му омогућава да ефикасно прати стање система, реагује на системске догађаје и одржава ажурним корисничко искуство.

Ова карактеристика је кључна за глатко функционисање покретача из следећих разлога:

- Континуитет рада - Сервис омогућава непрекидно функционисање у позадини, независно од животног циклуса активности или корисничког интерфејса.
- Висок приоритет у систему - Андроид третира сервисе као компоненте високог приоритета, смањујући вероватноћу њиховог гашења при ограниченим ресурсима.
- Независно извршавање - Сервиси могу радити без везивања за конкретну активност, омогућавајући управљање сложеним позадинским задацима.
- Ефикасно реаговање на системске догађаје - Сервис може ефикасно одговарати на broadcast-ове и системске промене, чак и када покретач није у првом плану.
- Аутоматско покретање - Сервиси се могу конфигурисати да се аутоматски покрену при подизању система, осигуравајући тренутну доступност функција покретача.
- Оптимизација ресурса - Сервиси су оптимизовани за дуготрајан рад са минималним утицајем на потрошњу батерије и меморије.

Испод ћемо навести пример како TeatroosService креира и управља преклапајућим прозорима. Принцип је исти за сваки прозор те ћемо овде дати пример само за палету послова:

```
class TeatroosService : Service() {  Zeljana +5 *
    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {  Zeljana +2 *
        instance = this
        createTaskbarOverlay( context: this)
        startAppTasksChecker()

        startForeground()
        registerReceivers()
        createAppMenuOverlay(visible = false)

        val modifierKeySettingsDao = ModifierKeysDatabase.getDatabase( context: this).modifierKeySettings()
        val modifierKeySettingsRepository = ModifierKeySettingsRepository(modifierKeySettingsDao)
        val modifierKeySettingsViewModel: ModifierKeySettingsViewModel by lazy {
            ModifierKeySettingsViewModel.
                ViewModelProvider.
                    provideModifierKeyViewModel(repository = modifierKeySettingsRepository)
        }
        modifierKeySettingsViewModel.loadModifierKey()

        return START_NOT_STICKY
    }
```

Слика 4: Креирање палете послова при покретању сервиса

```
class TeatroosService : Service() {  Zeljana +5

    fun createTaskbarOverlay(context: Context) {  Zeljana +2

        val params = WindowManager.LayoutParams(
            WindowManager.LayoutParams.MATCH_PARENT,
            WindowManager.LayoutParams.WRAP_CONTENT,
            WindowManager.LayoutParams.TYPE_APPLICATION_OVERLAY,
            WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE,
            PixelFormat.TRANSLUCENT
        )
        params.gravity = Gravity.BOTTOM
        params.type = WindowManager.LayoutParams.TYPE_SYSTEM_DIALOG

        val composeView = ComposeView( context: this)
        composeView.setContent {
            CustomTaskBar(
                appMenuButtonHeight = TASKBAR_HEIGHT,
                taskMenuViewModel,
                service: this@TeatroosService,
                context
            )
        }
        taskbarView = composeView

        val lifecycleOwner = TeatroosLifecycleOwner()
        lifecycleOwner.performRestore( savedState: null)
        lifecycleOwner.handleLifecycleEvent(Lifecycle.Event.ON_CREATE)
        ViewTreeLifecycleOwner.set(composeView, MainActivity.instance)

        composeView.setViewTreeSavedStateRegistryOwner(lifecycleOwner)

        val viewModelStore = ViewModelStore()
        ViewTreeViewModelStoreOwner.set(composeView) { viewModelStore }

        removeAllOverlays()
        windowManager.addView(taskbarView, params)
    }
```

Слика 5: Функција која креира преклапајући прозор палете послова

```
class TeatroosService : Service() {  Zeljana +5 *
    fun removeOverlay(view: ComposeView?) {  Veljko +1
        if (view == appMenuView) {
            // app menu requires different way of closing
            if (appMenuVisible)
                toggleDrawer()
        } else if (view != null && view.windowToken != null) {
            windowManager.removeView(view)
        }
        notificationsViewVisible = false
    }

    fun removeAllOverlays() {  Zeljana
        removeOverlays(
            appMenuView,
            taskMenuView,
            notificationView,
            volumeControlView,
            calendarAndClockView,
            rightClickMenuView,
            desktopRightClickMenuView,
            usersListView,
            taskbarRightClickMenuView,
            powerView,
            languagesView
        )
        notificationsViewVisible = false
        taskMenuVisible = false
        calendarVisible = false
        volumeControlVisible = false
        languagesViewVisible = false
        usersViewVisible = false
    }

    @SuppressLint("ClickableViewAccessibility")  Zeljana *
    fun closingOverlay(view: ComposeView?) {
        view?.setOnTouchListener { _, motionEvent ->
            if (motionEvent.rawY <= resources.displayMetrics.heightPixels -
                TASKBAR_HEIGHT_PX && motionEvent.action == MotionEvent.ACTION_OUTSIDE) {
                removeOverlay(view)
            }
            false
        }
```

Слика 6: Функције за уклањање преклапајућих прозора

На првој слици видимо како сервис креира палету послова приликом покретања сервиса, што је у овом случају чим се покрене покретач апликација. Ово у комбниацији са чињеницом да се преклапајући прозор палете послова не налази у функцији

removeAllOverlays() као и заставице FLAG_NOT_FOCUSABLE (палета не добија фокус, него га пропушта компонентама иза ње) и TYPE_SYSTEM_DIALOG, нам гарантује да ће палета послова увек бити видљива на екрану.

Такође видимо како TeatroosService креира преклапајући прозор и у њега пакује изглед овог елемента описаног у CustomTaskBar() composable функцији.

С обзиром да је принцип креирања преклапајућих прозора исти за сваки елемент, у наставку су описани само изгледи тј. composable функције свих елемената.

4.2 Позадниски екран

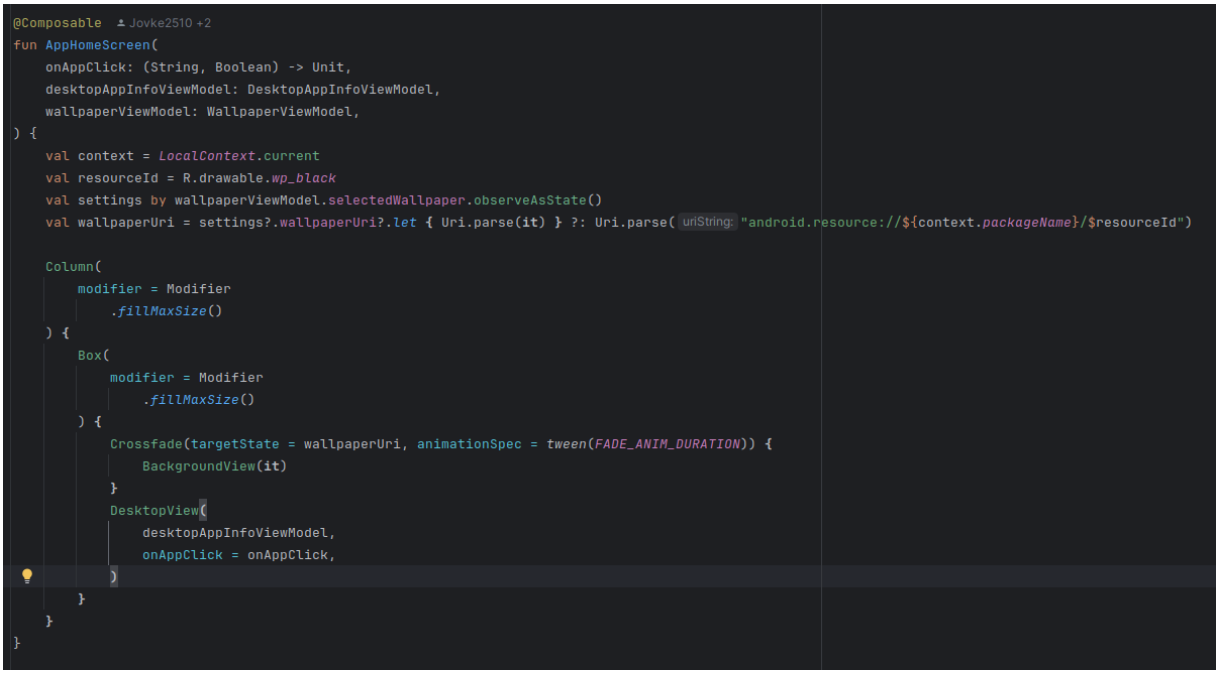
Позадински екран је описан помоћу **AppHomeScreen()** composable функције. Позива се из главне активности и представља највећи видљиви елемент покретача. Задужен је за приказ и интеракцију корисника са пречицама апликација као и приказ позадниске слике.

Параметри	Опис
onAppClick: (String, Boolean) → Unit	Функција задужена за покретање апликације приликом клика
desktopAppInfoViewModel: DesktopAppInfoViewModel	ViewModel задужен за добављање података везаних за пречице апликација које треба да буду приказане на екрану
WallpaperViewModel: WallpaperViewModel	ViewModel задужен за добављање података везаних за позадинску слику

Табела 1: Параметри AppHomeScreen() функције

AppHomeScreen() такође садржи две подфункције:

- BackgroundView(wallpaperUri: String) – служи за приказ позадинске слике
- DesktopView(desktopAppInfoViewModel, onAppClick) – задужена за приказ и управљање пречица апликација



Слика 7: AppHomeScreen() функција

На слици видимо распоред елемената који описују ове две подфункције. BackgroundView() (функција која описује поздаинску слику) и DesktopView() (описује пречице апликација) спаковане су у Box распоредну компоненту која дозвољава елементима унутар ње да буду постављени једни преко других, што за наш позадински екран значи да нам омогућава да поставимо позадину иза, а пречице апликација ставимо у први план без колизије између њих.

Такође можемо видети да се унутар ове функције добавља информација о тренутно постављеној позадини из базе помоћу wallpaperViewModel-а. Он прати тренутну вредност у бази, и уколико дође до промене, окида рекомпозицију AppHomeScreen() функције и поновно исцртавање на екран са новом вредношћу из базе. Ово омогућава AppHomeScreen() функцији да води рачуна само о приказивању тренутног стања, док проверу и добављање нових података обавља wallpaperViewModel. Слична ствар се дешава и унутар DesktopView() функције са њој прослеђеним desktopAppInfoViewModel-ом.

4.3 Палета послова (енгл. Taskbar)

Палета послова је описана помоћу **CustomTaskBar()** composable функције. Задужена је за приказ дугмади за почетни мени, брза подешавања, календар и подешавања звука, као и за приказ тренутно покренутих апликација.

Параметри	Опис
appMenuButtonHeight: Dp	Променљива која задаје висину палете послова
taskMenuModel: TaskMenuViewModel	ViewModel задужен за добављање података везаних за тренутно отворене апликације
service: TeatroosService	Сервис задужен за управљањем преклапајућим прозорима менија
context: Context	Контекст променљива која говори за коју активност је везана функција

Табела 2: Параметри CustomTaskBar() функције

Палета послова подељена је логички на леву, централну и десну целину које су задужене за приказ кореспондирајућиш елемената на основу њихове позиције у предложеном дизајну. Тако CustomTaskBar() функција садржи три подфункције:

- LeftTaskBarClickables(context, service, Modifier.align(Alignment.CenterStart)) – функција која приказује елементе у левој целини палете послова (дугмад за почетни мени, брзу претрагу, мени задатака (енгл. taskmenu))
- RunningApps(taskMenuModel, Modifier.align(Alignment.Center)) – composable функција која приказује тренутно покренуте апликације и индикатор тренутно фокусиране апликације
- RightTaskBarClickables(context, service, Modifier.align(Alignment.CenterEnd), selectedLanguagesViewModel) – composable функција која приказује елементе у десној целини палете послова (дугмад за брза подешавања, звук, језик, календар)

```
@Composable  Veljko Jovanovic +3
fun CustomTaskBar(
    appMenuButtonHeight: Dp,
    taskMenuModel: TaskMenuViewModel,
    service: TeatroosService,
    context: Context
) {
    val selectedLanguageDao = SelectedLanguagesDatabase.getDatabase(context).selectedLanguages()
    val selectedLanguageRepository = SelectedLanguageRepository(selectedLanguageDao)
    val selectedLanguagesViewModel: SelectedLanguagesViewModel by lazy {
        SelectedLanguagesViewModel.ViewModelProvider.provideSelectedLanguagesViewModel(repository = selectedLanguageRepository)
    }

    Box(
        modifier = Modifier
            .teatroBackground( angle: 180f)
            .height(appMenuButtonHeight)
            .fillMaxWidth()
            .padding(TASKBAR_INNER_CONTENT_PADDING)
    ) {
        LeftTaskbarClickables(
            context = context,
            service = service,
            modifier = Modifier.align(Alignment.CenterStart)
        )
        RunningApps(taskMenuModel, modifier = Modifier.align(Alignment.Center))
        RightTaskbarClickables(
            context = context,
            service = service,
            modifier = Modifier.align(Alignment.CenterEnd),
            selectedLanguagesViewModel
        )
    }
}
```

Слика 8: CustomTaskBar() функција

На слици видимо да је палета послова реализована кроз Row распоређивачки елемент, што и не изненађује толико с обзиром да је палета послова једна хоризонтална трака која пакује елементе једне поред других.

4.4 Почетни мени

Почетни мени је описан помоћу **AppMenu()** composable функције. Задужен је за приказивање свих инсталираних апликација на уређају, скоро отворених апликација и омогућава претрагу апликација, фајлова и интернета.

Параметри	Опис
service: TeatroosService	Сервис задужен за управљањем преклапајућим прозорима менија

Табела 3: Параметри AppMenu() функције

AppMenu() функција такође садржи следеће подфукнције:

- AppSearchBar() - функција која је задужена за претрагу инсталираних апликација

- RecentAppList(APP_DRAWER_WIDTH, service) – функцкија која приказује скоро отворене апликације
- AppList(service) – функција која приказује све инсталиране апликације на уређају
- FileSearch(service) – функција која претражује унети упит унутар фајлова
- WebSearch(service) - функција која претражује унети упит путем интернета

```
@Composable  Veljko +4
fun AppMenu(service: TeatroosService) {

    val appSearchViewModel: AppSearchViewModel by lazy {
        AppSearchViewModel.ViewModelProvider.provideAppSearchViewModel()
    }
    val searchText by appSearchViewModel.searchText.collectAsState()

    Column(
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = Modifier
            .clip(RoundedCornerShape(ROUNDED_CORNER_SHAPE_AMOUNT))
            .padding(OVERLAY_SHADOW_WIDTH) // for shadow
            .shadow(
                elevation = OVERLAY_SHADOW_WIDTH,
                shape = RoundedCornerShape(OVERLAY_ROUNDED_CORNERS_RADIUS),
            )
            .teatroBackground( angle: 160f)
            .fillMaxSize()
            .padding(top = APP_DRAWER_INNER_CONTENT_PADDING)
    ) {
        AppSearchBar()
        Column(
            modifier = Modifier
                .padding(
                    start = APP_DRAWER_INNER_CONTENT_PADDING,
                    end = APP_DRAWER_INNER_CONTENT_PADDING,
                    bottom = APP_DRAWER_INNER_CONTENT_PADDING
                )
                .fillMaxSize()
        ) {
            Spacer(modifier = Modifier.height(SPACER_HEIGHT))
            if (searchText.isEmpty()) {
                Text(
                    text = stringResource("Recently used"),
                    color = getContentColor(),
                    modifier = Modifier.padding(vertical = MODIFIER_PADDING_20),
                    fontSize = MaterialTheme.typography.button.fontSize,
                )
```

Слика 9: AppMenu() функција

```
fun AppMenu(service: TeatroosService) {
    ) {
                .fillMaxSize()
        ) {
            Spacer(modifier = Modifier.height(SPACER_HEIGHT))
            if (searchText.isEmpty()) {
                Text(
                    text = stringResource("Recently used"),
                    color = getContentColor(),
                    modifier = Modifier.padding(vertical = MODIFIER_PADDING_20),
                    fontSize = MaterialTheme.typography.button.fontSize,
                )
                RecentAppList(APP_DRAWER_WIDTH, service)
                Divider(
                    thickness = DIVIDER_THICKNESS,
                    modifier = Modifier.padding(vertical = APP_DRAWER_SECTION_SPACER_PADDING),
                    color = getElementInactiveColor()
                )
                AppList(service)

            } else {
                Spacer(modifier = Modifier.padding(APP_DRAWER_SECTION_SPACER_PADDING))
                AppList(service)
            }
            Spacer(modifier = Modifier.weight(1f))

            if (searchText.isNotEmpty()) {
                FilesSearch(service)
                Spacer(modifier = Modifier.padding(vertical = APP_DRAWER_SEARCH_OPTION_SPACER))
                WebSearch(service)
            }
        }

    }

}
```

Слика 10: AppMenu() функција

Ако погледамо структуру кода почетног менија, иако делује комплексно на први поглед, у ствари је веома једноставна. Састоји се из главног Column распоређивачког елемента, који своје елементе распоређује вертикално, и затим свих осталих целина које се ређају једна испод друге (прво трака претраге па испод ње следећу колону у коју се пакују вертикално њени елементи итд.). Оваква структура не само да је лако читљива већ је и лако скалабилна (могли би смо додати нови графички елемент било где у структури без већих проблема), у чему се управо и огледа моћ JetpackCompose алата.

4.5 Брза подешавања

Брза подешавања су описана помоћу **NotificationsAndSettingsWindow()** composable функције. Задужена су за приказ пречица за подешавања као и за приказ обавештења.

Параметри	Опис
service: TeatroosService	Сервис задужен за управљањем преклапајућим прозорима менија
context: Context	Контекст променљива која говори за коју активност је везана функција

Табела 4: Параметри NotificationAndSettingsWindow() функције

NotificationsAndSettingsWindow() функција подељена је на више подфункција:

- TeatroCircledIconButton() - функција за округлу прилагођену дугмад за брза подешавања
- TeatroRoundedCornerTextButton() - функција за прилагођену уоквирену дугмад за брза подешавања
- QuickSettings() - функција за приказ брзих подешавања
- Notifications(notificationViewModel, service) – функција за приказ нотификација

```
@Composable  Veljko +4
fun NotificationAndSettingsWindow(service: TeatroosService, context: Context) {
    val notificationViewModel: NotificationViewModel by lazy {
        NotificationViewModel.ViewModelProvider.provideNotificationViewModel()
    }
    val powerOffText = stringResource(id = "Are you sure you want to power off the device?")
    val rebootText = stringResource(id = "Are you sure you want to restart the device?")

    Column(
        modifier = Modifier
            .clip(RoundedCornerShape(OVERLAY_ROUNDED_CORNERS_RADIUS))
            .padding(OVERLAY_SHADOW_WIDTH) // for shadow
            .shadow(
                elevation = OVERLAY_SHADOW_WIDTH,
                shape = RoundedCornerShape(OVERLAY_ROUNDED_CORNERS_RADIUS),
            )
            .teatroBackground( angle: 160f)
            .fillMaxSize()
            .padding(NAS_OVERLAY_INNER_PADDING) // for inner content

    ) {
        Row(
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.spacedBy(NAS_HEADER_PADDING_BETWEEN_CONTENT),
            modifier = Modifier
                .fillMaxWidth()
                .padding(vertical = NAS_HEADER_PADDING)
        ) {...}
        Divider(
            thickness = DIVIDER_THICKNESS,
            color = getElementInactiveColor(),
            modifier = Modifier
                .fillMaxWidth()
                .padding(vertical = NAS_HEADER_PADDING)
        )
        Text(
            text = stringResource("Quick settings"),
            color = getContentColor(),
            modifier = Modifier.padding(vertical = NAS_SECTION_TITLE_PADDING)
        )/* Quick settings tiles */
        QuickSettings()
        Divider(
            thickness = DIVIDER_THICKNESS,
            color = getElementInactiveColor(),
            modifier = Modifier
                .fillMaxWidth()
                .padding(vertical = NAS_SECTION_TITLE_PADDING)
        )
        Notifications(notificationViewModel, service)
    }
}
```

Слика 11: NotificationAndSettingsWindow() функција

Као и код почетног менија, иако наизглед компликована, структура графичког елемента за брза подешавања је само скуп мноштва једноставних мањих елемената и различитих распоређивачких компоненти.

4.6 Taskmenu

Taskmenu је описан помоћу **TaskMenuView()** composable функције. Задужен је за приказ и управљање тренутно отвореним апликацијама.

Параметри	Опис
taskViewModel: TaskMenuViewModel	ViewModel задужен за добављање података везаних за тренутно отворене апликације
onAppClick: (String, Boolean) → Unit	Функција задужена за покретање апликације приликом клика
service: TeatroosService	Сервис задужен за управљањем преклапајућим прозорима менијаСервис задужен за управљањем преклапајућим прозорима менија

Табела 5: Параметри TaskMenuView() функције

TaskMenuView() функција има само једну подфункцију:

- TaskView(taskViewModel, app, taskViewHeight, onAppClick, service, initialFocusRequester, isFirstTask) – задужена за приказ сваког прозора отворених апликација

```kotlin
/* The box in which the tasks themselves are placed. */
@OptIn(ExperimentalPagerApi::class)  Zeljana +2
@Composable
fun TaskMenuView(
    taskViewModel: TaskMenuViewModel,
    onAppClick: (String, Boolean) -> Unit,
    service: TeatroosService
) {

    val TASKS_PER_PAGE = TASK_COLUMNS_PER_PAGE * TASK_ROWS_PER_PAGE
    val taskList = taskViewModel.openedApps.collectAsState()

    // open new page on number of opened tasks -1 (to prevent adding new empty page when we have max number of tasks on current page)
    // and +1 because there is always at least one page, even if empty
    val pagesNum = (taskList.value.size - 1) / TASKS_PER_PAGE + 1
    val pagerState = rememberPagerState()
    val scope = rememberCoroutineScope()
    val initialFocusRequester = remember { FocusRequester() }


    BoxWithConstraints(
        modifier = Modifier
            .padding(OVERLAY_SHADOW_WIDTH) // for shadow
            .shadow(
                elevation = OVERLAY_SHADOW_WIDTH,
                shape = RoundedCornerShape(OVERLAY_ROUNDED_CORNERS_RADIUS),
            )
            .teatroBackground( angle: 180f)
            .size(
                width = TASKMENU_VIEW_WIDTH, height = TASKMENU_VIEW_HEIGHT
            ) // @todo configure to be dynamic
    ) {
        HorizontalPager(
            count = pagesNum,
            state = pagerState,
            modifier = Modifier
                .fillMaxSize()
        ) { pageIndex ->
            LazyVerticalGrid(
                columns = GridCells.Fixed(TASK_COLUMNS_PER_PAGE),
                modifier = Modifier
                    .fillMaxSize(),
                contentPadding = PaddingValues(TASKMENU_CONTENTS_PADDING),
                verticalArrangement = Arrangement.spacedBy(TASK_GRID_PADDING),
                horizontalArrangement = Arrangement.spacedBy(TASK_GRID_PADDING)
            ) {
                for (i in 0 ≤ until  < TASKS_PER_PAGE) {

                    val index = (pageIndex * TASKS_PER_PAGE) + i

                    // prevents IOB exception when last page has less items than the maximum supported
                    if (index < taskList.value.size) {
                        val app = taskList.value[index]

                        item {
                            val isFirstTask = index == 0 && pageIndex == 0
```

Слика 12: TaskMenuView() функција

```
fun TaskMenuView(
    ) {
        ) { pageIndex ->
            ) {
                        item {
                            TaskView(
                                taskViewModel,
                                app,
                                maxHeight,
                                onAppClick,
                                service,
                                if (isFirstTask) initialFocusRequester else null,
                                isFirstTask
                            )
                        }
                    }
                }
            }
        }
        Row(
            modifier = Modifier
                .align(Alignment.BottomCenter)
                .offset(y = PAGE_INDICATOR_ROW_Y_OFFSET)
        ) {
            // @todo replace these boxes with proper icons when we acquire the ui icon pack
            repeat(pagesNum) { page ->
                Box(
                    modifier = Modifier
                        .clip(CircleShape)
                        .pointerInput(Unit) {
                            detectTapGestures(
                                onTap = {
                                    scope.launch {
                                        pagerState.scrollToPage(page)
                                    }
                                }
                            )
                        }
                        .border(
                            color = getElementInactiveColor(),
                            width = TASK_BORDER_WIDTH,
                            shape = CircleShape
                        )
                        .background(
                            color = if (page == pagerState.currentPage) {
                                getContentColor()
                            } else {
                                getElementInactiveColor()
                            }
                        )
                        .size(PAGE_INDICATOR_SIZE)
                        .padding(PAGE_INDICATOR_PADDING)
                )
            }
        }
    }
}
```

Слика 13: TaskMenuView() функција

4.7 Контекст мени на десни клик

Контекст мени на десни клик је описан помоћу три функције, у зависности од елемента над којим се прави. Уколико се мени прави десним кликом на десктоп онда се користи функција **DesktopRightClickMenu()**, уколико се прави десним кликом на палету послова **TaskbarRightClickMenu()** и уколико се прави десним кликом на иконице апликација користи се **RightClickMenu()**. Свака од ових функција имају исту улогу а то је исцртавање контекст менија са додатним опцијама.

Параметри	Опис
service: TeatroosService	Сервис задужен за управљањем преклапајућим прозорима менијаСервис задужен за управљањем преклапајућим прозорима менија

Табела 6: Параметри DesktopRightClickMenu() функције

Параметри	Опис
service: TeatroosService	Сервис задужен за управљањем преклапајућим прозорима менијаСервис задужен за управљањем преклапајућим прозорима менија
app: App	Апликација којом се управља
context: Context	Контекст променљива која говори за коју активност је везана функција

Табела 7: Параметри TaskbarRightClickMenu() функције

Параметри	Опис
service: TeatroosService	Сервис задужен за управљањем преклапајућим прозорима менијаСервис задужен за управљањем преклапајућим прозорима менија
app: DesktopAppInfo	Пречица апликације којом се управља

Табела 8: Параметри RightClickMenu() функције

```
@Composable  Jovke2510 +2
fun DesktopRightClickMenu(service: TeatroosService) {
    val context = LocalContext.current
    Column (
        modifier = Modifier
            .clip(RoundedCornerShape(ROUNDED_CORNER_SHAPE_AMOUNT))
            .padding(OVERLAY_SHADOW_WIDTH) // for shadow
            .shadow(
                elevation = OVERLAY_SHADOW_WIDTH,
                shape = RoundedCornerShape(OVERLAY_ROUNDED_CORNERS_RADIUS),
            )
            .background(getElementInactiveColor())
            .wrapContentHeight()
            .width(RIGHT_CLICK_MENU_WIDTH)
            .padding(MODIFIER_PADDING_5),
        verticalArrangement = Arrangement.spacedBy(MODIFIER_PADDING_5)
    ) {
        Row(
            modifier = Modifier
                .clickable {
                    launchSettings(context = context, service = service)
                }
                .fillMaxWidth()
                .padding(MODIFIER_PADDING_5, MODIFIER_PADDING_0)
        ) {
            Text(
                text = stringResource("Personalization"),
                color = getContentColor()
            )
        }
    }
}
```

Слика 14: DesktopRightClickMenu() функција

```
@Composable  Zeljana +2
fun TaskbarRightClickMenu(service: TeatroosService, app: App, context: Context) {
    val taskId = Util.getApplicationTaskId(context, app.packageName)
    val taskViewModel = TaskMenuViewModel.ViewModelProvider.provideTaskMenuViewModel()
    val exitFullscreen = stringResource(id = "Press F11 to exit fullscreen.")
    Column(
        modifier = Modifier
            .clip(RoundedCornerShape(ROUNDED_CORNER_SHAPE_AMOUNT))
            .background(
                color = getElementInactiveColor(),
            )
            .wrapContentHeight()
            .width(RIGHT_CLICK_MENU_WIDTH)
            .padding(MODIFIER_PADDING_5),
        verticalArrangement = Arrangement.spacedBy(MODIFIER_PADDING_5)
    ) {
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .padding(MODIFIER_PADDING_5, MODIFIER_PADDING_0)
                .clickable {
                    service.removeOverlay(service.taskbarRightClickMenuView)
                    val aM = context.getSystemService(Context.ACTIVITY_SERVICE) as ActivityManager
                    aM.moveTaskToFront(taskId, ActivityManager.MOVE_TASK_NO_USER_ACTION)
                    Util.setWindowingMode(context, taskId,  windowingMode: 1)
                    service.taskbarView?.visibility = View.INVISIBLE
                    Toast
                        .makeText(context, exitFullscreen, Toast.LENGTH_LONG)
                        .show()
                }
        ) {
            Text(
                text = stringResource(id = "Fullscreen"),
                color = getContentColor()
            )
        }
        Row(
            modifier = Modifier
                .clip(RoundedCornerShape(ROUNDED_CORNER_SHAPE_AMOUNT))
                .fillMaxWidth()
                .padding(MODIFIER_PADDING_5, MODIFIER_PADDING_0)
```

Слика 15: TaskbarRightClickMenu() функција

```
    ) {
                .fillMaxWidth()
                .padding(MODIFIER_PADDING_5, MODIFIER_PADDING_0)
                .clickable {
                    service.removeOverlay(service.taskbarRightClickMenuView)
                    val intent = Intent( action: "android.intent.action.CAPTION_BUTTON")
                    intent.putExtra( name: "type",  value: "maximize")
                    intent.putExtra( name: "task_id", taskId)
                    context.sendBroadcast(intent)
                }
        ) {
            if (Util.isWindowMaximized(context, taskId)) {
                Text(
                    text = stringResource(id = "Minimize"),
                    color = getContentColor()
                )
            } else {
                Text(
                    text = stringResource(id = "Maximize"),
                    color = getContentColor()
                )
            }
        }
        Row(
            modifier = Modifier
                .clip(RoundedCornerShape(ROUNDED_CORNER_SHAPE_AMOUNT))
                .fillMaxWidth()
                .padding(MODIFIER_PADDING_5, MODIFIER_PADDING_0)
                .clickable {
                    //implement quiting settings
                    Util.closeApp(context, app.packageName)
                    taskViewModel.removeApp(app) // to notify both taskbar and taskmenu to recompose
                    service.removeOverlay(service.taskbarRightClickMenuView)
                }
        ) {
            Text(
                text = stringResource(id = "Quit"),
                color = getContentColor()
            )
        }
    }
}
```

Слика 16: TaskbarRightClickMenu() функција

```
@OptIn(DelicateCoroutinesApi::class)  Jovke2510 +3
@Composable
fun RightClickMenu(service: TeatroosService, app: DesktopAppInfo) {
    val context = LocalContext.current
    val desktopAppInfoViewModel =
        viewModel<DesktopAppInfoViewModel>(MainActivity.instance as ViewModelStoreOwner)
    val appList =
        desktopAppInfoViewModel.getDesktopAppInfoList().collectAsState(initial = emptyList()).value
    val text: String
    val addRemoveSwitch: Int
    val existsInDatabase by desktopAppInfoViewModel.existsInDatabase.collectAsState()
    val appSearchViewModel = AppSearchViewModel.ViewModelProvider.provideAppSearchViewModel()

    desktopAppInfoViewModel.doesItemExist(app.packageName)

    if (existsInDatabase) {
        text = stringResource("Remove shortcut")
        addRemoveSwitch = -1
    } else {
        text = stringResource("Add shortcut")
        addRemoveSwitch = 1
    }

    Column(
        modifier = Modifier
            .clip(RoundedCornerShape(ROUNDED_CORNER_SHAPE_AMOUNT))
            .padding(OVERLAY_SHADOW_WIDTH) // for shadow
            .shadow(
                elevation = OVERLAY_SHADOW_WIDTH,
                shape = RoundedCornerShape(OVERLAY_ROUNDED_CORNERS_RADIUS),
            )
            .background(color = getElementInactiveColor())
            .wrapContentHeight()
            .width(RIGHT_CLICK_MENU_WIDTH)
            .padding(MODIFIER_PADDING_5),
        verticalArrangement = Arrangement.spacedBy(MODIFIER_PADDING_5)
    ) {
        Row(modifier = Modifier
            .fillMaxWidth()
            .padding(MODIFIER_PADDING_10, MODIFIER_PADDING_0)
            .clickable {
                GlobalScope.launch {
```

Слика 17: RightClickMenu() функција

```
    ) {
            .clickable {
                GlobalScope.launch {
                    when (addRemoveSwitch) {
                        -1 -> {...}

                        1 -> {...}

                        else -> Toast
                            .makeText(
                                context,
                                "Something went wrong with menu",
                                Toast.LENGTH_SHORT
                            )
                            .show()
                    }
                    service.removeOverlay(service.rightClickMenuView)
                }
            }) {
            Text(
                text = text, color = getContentColor()
            )
        }
        if (app.applicationType != APP_TYPE_WEB_SHORTCUT && app.packageName != getApp(
                LocalContext.current,
                TEATRO_BROWSER_PACKAGE_NAME
            )?.packageName
        ) {
            Row(modifier = Modifier
                .fillMaxWidth()
                .padding(MODIFIER_PADDING_10, MODIFIER_PADDING_0)
                .clickable {
                    service.removeAllOverlays()
                    uninstallApp(app.packageName)
                }
            ) {
                Text(text = stringResource("Uninstall"), color = getContentColor())
            }
        }
    }
}
```

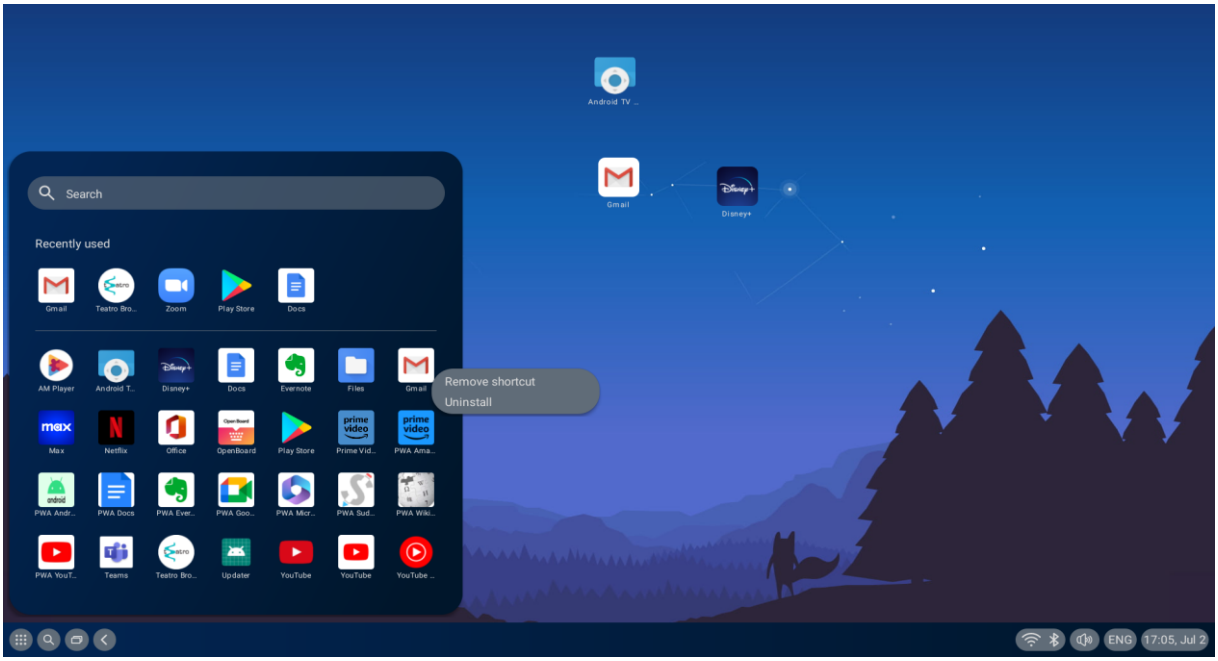
Слика 18: RightClickMenu() функција

5. Резултати

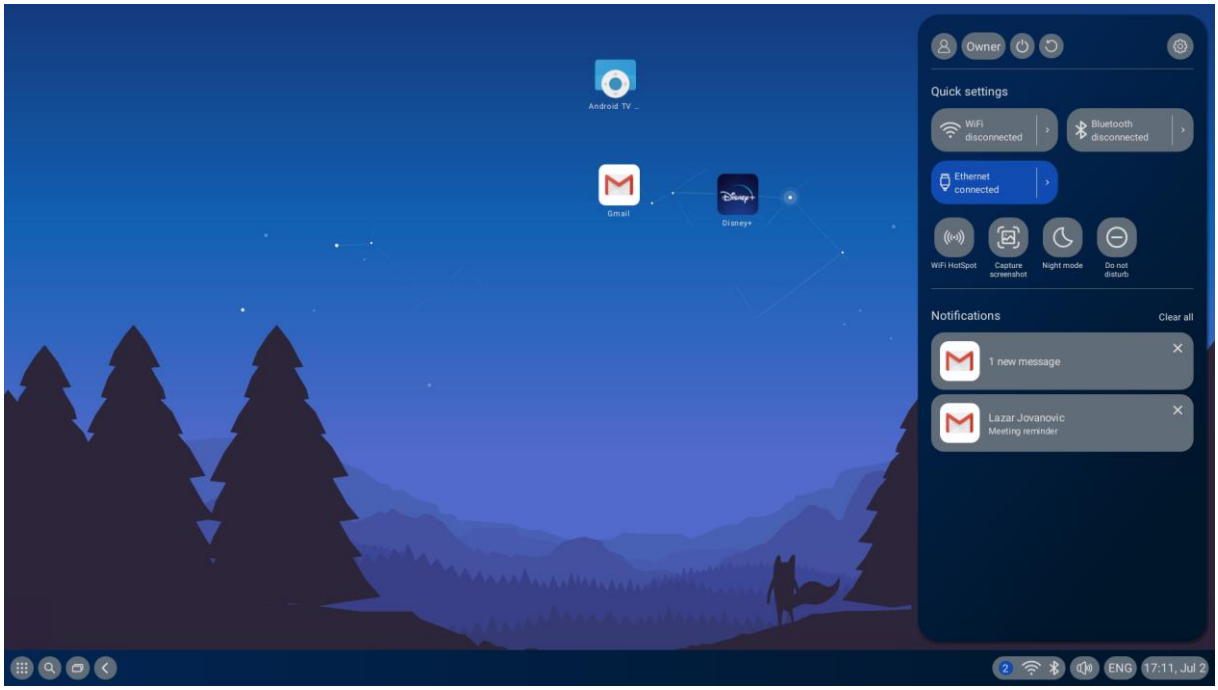
У овом поглављу ће бити описан процес тестирања и верификације свих графичких елемената који чине наш налик-на-десктоп покретач апликација. Потребно испитивање се односи на сам графички изглед и лакоћу коришћења корисничке спреге а потом и на реакцију приликом притиска на одређену визуелну компоненту.

5.1 Изглед корисничке спреге

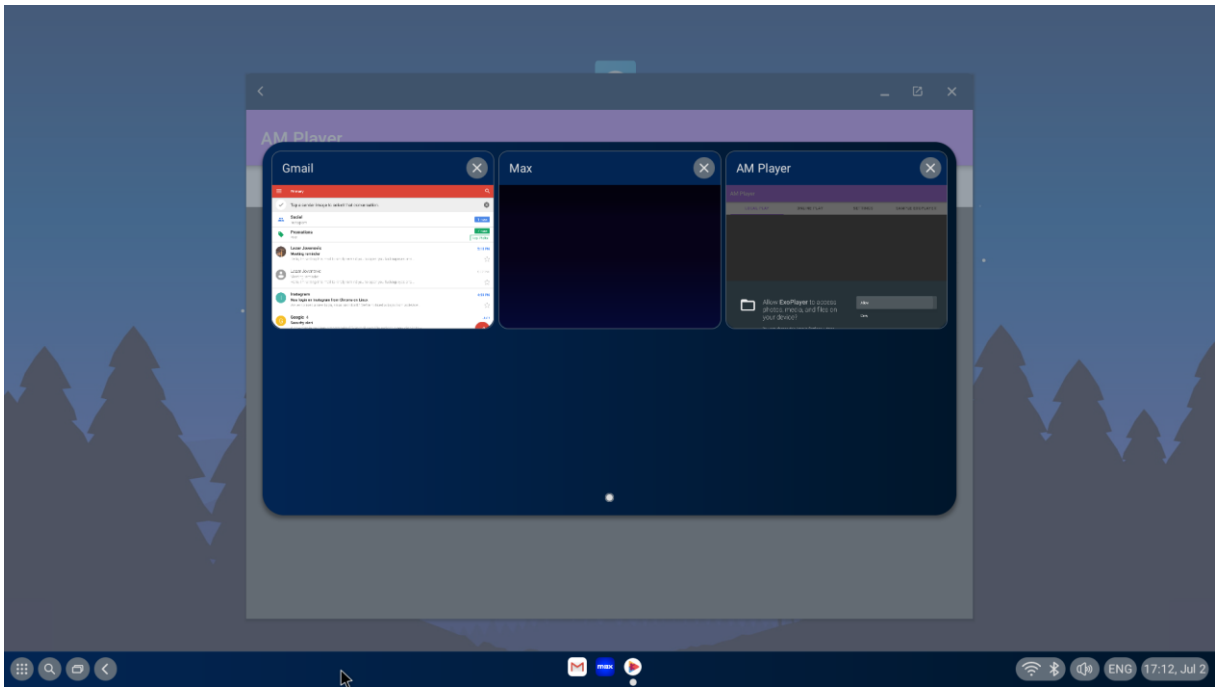
Након завршене имплементације добили смо коначан изглед корисничке спреге нашег покретача:



Слика 19: Изглед корисничке спреге (палета послова, почетни мени, контекстни мени)



Слика 21: Изглед корисничке спреге (брза подешавања, пречице апликција)



Слика 20: Изглед корисничке спреге (taskmenu, тренутно покренуте апликације)

5.2 Тестирање функционалности спреге

Након потврде да се изглед корисничке спрегекао поклапа са предложеним дизајном, спроведено је свобухватно тестирање корисничке спреге.

Почетна фаза тестирања била је усмерена на постојаност палете послова на екрану, кључни елемент спреге. Утврђено је да је палета константно присутна на екрану, обезбеђујући непрекидан приступ основним функцијама корисничке спреге. Тестирање интеракције са дугмадима на палети потврдило је да сваки притисак успешно отвара одговарајући мени. Посебна пажња посвећена је понашању менија при отварању, где је верификовано да се сви претходно отворени менији аутоматски затварају осим палете послова. Ово осигурава прегледност корисничке спреге и спречава нагомилавање отворених елемената на екрану.

Даље тестирање обухватило је проверу затварања активног менија, где је потврђено да се мени може затворити једноставним кликом ван његовог оквира, што доприноси интуитивности корисничкох искуства. Значајан део тестирања био је посвећен управљању пречицама апликација. Верификовано је да се пречице могу додавати, на екран без колизије са већ постављеним пречицама, ефикасно уклањати када нису потребне као и померати по екрану по жељи. Потврђено је да спрега спречава преклапање пречица и враћа их на оригиналну позицију приликом колизије са другим пречицама.

Завршна фаза тестирања фокусирала се на функционалност покретања апликација. Потврђено је да се апликције успешно покрећу како кликом на пречице на екрану, тако и кроз почетни мени, обезбеђујући корисницима вишеструке начине приступа жељеним програмима.

Важно је напоменути да су сви ови тестови спроведени на различитим величинама екрана, укључујући стандардне мониторе и велике ТВ екране. Резултати су показали да корисничка спрега задржава своју функционалност и естетику на свим тестираним величинама екрана, потврђујући његову прилагодљивост коју смо желели.

6. Закључак

У овом раду имплементирана је налик-на-десктоп графичка корисничка спрега покретача апликација за Андроид платформу.

У раду су описани концепт решења и дизајн шаблон по којем је рађена корисничка спрега, програмска имплементација дизајн шаблона коришћењем Kotlin програмског језика и JetpackCompose алата као и верификација функционалности решења и подударности са предложеним дизајном.
Приликом имплементације посебна пажња посвећена је прилагодљивости и преносивости овог решења, како би омогућили беспрекоран рад и изглед на различитим величинама монитора и телевизора.
Праваци даљег развоја могу укључивати побољшање корисничког искуства, измене у дизајну или додавање нових функционалности. Једна од функционалности која би могла бити имплементирана је додатавање контекст менија на десни клик на насловној траци прозора апликација (енгл. caption bar). Овај мени би могао да пружа велики број додатних функционалности самом покретачу попут опција за управљање прозором (минимизуј, максимизуј, сакриј, затвори) или опције за снимак екрана прозора.

Закључно, налик-на-десктоп корисничка спрега омогућава кориснику да користи свој Андроид уређај на један нов а интуитиван начин, обједињујући функционалности класничних десктоп рачунара и Андроид уређаја у јединствено решење и редефинише начине размишљања о коришћењу и могућностима Андроид платформе.

7. Литература

[1]……Harun Wangereka: “Mastering Kotlin for Android 14: Build powerful Android apps from scratch using Jetpack libraries and Jetpack Compose “, Април 2024.

[2]……Neil Smyth: “Jetpack Compose 1.5 Essentials Developing Android Apps with Jetpack Compose 1.5, Android Studio, and Kotlin”, Јануар 2024.

[3]…….https://developer.android.com/develop/ui/compose/documentation, посећено 06.06.2024.

[4]……. https://developer.android.com/develop, посећено 06.06.2024.

[5]……. https://kotlinlang.org/, посећено 06.06.2024.